Specifications for Managed Strings

Hal Burch
CERT/CC
Software Engineering Institute

Fred Long
Department of Computer Science
University of Wales, Aberystwyth

Robert Seacord
CERT/CC
Software Engineering Institute

May 2006

TECHNICAL REPORT
CMU/SEI-2006-TR-006
ESC-TR-2006-006

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Specifications for Managed Strings

CMU/SEI-2006-TR-006
ESC-TR-2006-006

Hal Burch, CERT/CC, Software Engineering Institute

Fred Long, Department of Computer Science,
University of Wales, Aberystwyth

Robert Seacord, CERT/CC, Software Engineering Institute

May 2006

Networked Systems Survivability Program

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

Abstract

This report describes a managed string library for the C programming language. Many software vulnerabilities in C programs result from the misuse of standard C string manipulation functions. Programming errors common to string manipulation logic include buffer overflow, truncation errors, string termination errors, and improper data sanitation. The managed string library provides mechanisms to eliminate or mitigate these problems and improve system security. A proof-of-concept implementation of the managed string library is available from the Secure Coding area of the CERT Web site.

1 Introduction

1.1 String Manipulation Errors

Many software vulnerabilities in C programs arise through the use of the standard C string manipulating functions. String manipulation programming errors include buffer overflow through string copying, truncation errors, termination errors and improper data sanitization.

Buffer overflow can easily occur during string copying if the fixed-length destination of the copy is not large enough to accommodate the source of the string. This is a particular problem when the source is user input, which is potentially unbounded. The usual programming practice is to allocate a character array that is generally large enough. However, this fixed-length array can still be exploited by a malicious user who supplies a carefully crafted string that overflows the array in such a way that the security of the system is compromised. This remains the most common exploit in fielded C code today.

In attempting to overcome the buffer overflow problem, some programmers limit the number of characters that are copied. This can result in strings being improperly truncated, which in turn results in a loss of data that may lead to a different type of software vulnerability.

A special case of truncation error is a termination error. Many of the standard C string functions rely on strings being null terminated. However, the length of a string does not include the null character. If just the non-null characters of a string are copied, the resulting string may not be properly terminated. A subsequent access may run off the end of the string, corrupting data that should not have been touched.

Finally, inadequate data sanitization can also lead to software vulnerabilities. In order to properly function, many applications require that data not contain certain characters. Ensuring that the strings used by the application do not include illegal characters can often prevent malicious users from exploiting an application.

1.2 Proposed Solution

A secure string library should provide facilities to guard against the programming errors described above. Furthermore, it should satisfy the following requirements:

1. Operations should succeed or fail unequivocally.

2. The facilities should be familiar to C programmers so that they can easily be adopted and existing code easily converted.
3. There should be no surprises in using the facilities. The new facilities should have similar semantics to the standard C string manipulating functions. Again, this will help with the conversion of legacy code.

Of course, some compromises are needed to meet these requirements. For example, it is not possible to completely preserve the existing semantics and provide protection against the programming errors described above.

Libraries that provide string manipulation functions can be categorized as static or dynamic. Static libraries rely on fixed-length arrays. A static approach cannot easily overcome the problems described. With a dynamic approach, strings are resized as necessary. This approach can more easily solve the problems, but a consequence is that memory can be exhausted if input is not limited. To mitigate this problem, the managed string library supports an implementation-defined maximum string length. The minimum system-defined maximum string length for a conforming implementation is **BUFSIZ-1** (see [ISO/IEC:99, Section 7.19.2]). Additionally, the string creation function allows for the specification of a per string maximum length.

1.3 The Managed String Library

This managed string library was developed in response to the need for a string library that could improve the quality and security of newly developed C language programs while eliminating obstacles to widespread adoption and possible standardization.

The managed string library is based on a dynamic approach in that memory is allocated and reallocated as required. This approach eliminates the possibility of unbounded copies, null-termination errors, and truncation by ensuring there is always adequate space available for the resulting string (including the terminating null character).

A runtime-constraint violation occurs when memory cannot be allocated. In this way, the managed string library accomplishes the goal of succeeding or failing loudly.

The managed string library also provides a mechanism for dealing with data sanitization by (optionally) checking that all characters in a string belong to a predefined set of “safe” characters.

1.4 Wide Character and Null-Terminated Byte Strings

A number of managed string functions accept either a null-terminated byte string or a wide character string as input or provide one of these string types as a return value. The managed string library works equally well with either type of string. For example, it is possible to create a managed string from a wide character string and then extract a null-terminated byte string (or vice versa). It is also possible to copy a null-terminated byte string and then concatenate a wide character string. Managed string functions will handle conversions implicitly when possible. If a conversion cannot be performed, the operation is halted and a runtime-constraint error reported.

Strings are maintained in the format in which they are initially provided, until such a time that a conversion is necessary. String promotions are relatively simple: performing an operation on two null-terminated byte strings results in a null-terminated byte string. An operation on a null-terminated byte string and a wide character string results in a wide character string. Operations on two wide character strings results in a wide character string. Conversions are performed as necessary in the locale defined at the time the conversion occurs.

Managed strings also support the definition of a restricted character set that identifies the set of allowable characters for the string. When an operation requires that a null-terminated byte string be converted to a wide character string, the restricted character set is also converted as part of the operation.

2 Library

2.1 Use of errno

An implementation may set **errno** for the functions defined in this technical report but is not required to do so.

2.2 Runtime-Constraint Violations

Most functions in this technical report include as part of their specifications a list of runtime-constraints, which are requirements on the program using the library. Despite its name, a runtime-constraint is not a kind of constraint. Implementations shall verify that the runtime-constraints for a library function are not violated by the program.

Implementations shall check that the runtime-constraints specified for a function are met by the program. If a runtime-constraint is violated, the implementation shall call the currently registered constraint handler (see **set_constraint_handler** in Section 2.7, General Utilities <stdlib.h>). Multiple runtime-constraint violations in the same call to a library function result in only one call to the constraint handler. It is unspecified which one of the multiple runtime-constraint violations cause the handler to be called.

Sometimes, the runtime-constraints section for a function states an action to be performed if a runtime-constraint violation occurs. Such actions are performed before calling the runtime-constraint handler. Sometimes, the runtime-constraints section lists actions that are prohibited if a runtime-constraint violation occurs. Such actions are prohibited to the function both before calling the handler and after the handler returns.

The runtime-constraint handler might not return. If it does, the library function whose runtime-constraint was violated shall return some indication of failure as given by the returns section in the function's specification.

Although runtime-constraints replace many cases of undefined behavior from International Standard ISO/IEC 9899:1999 [ISO/IEC 99], undefined behavior can still occur. Implementations are free to detect any case of undefined behavior and treat it as a runtime-constraint violation by calling the runtime-constraint handler. This license comes directly from the definition of undefined behavior.

2.3 Errors <errno.h>

The header **`<errno.h>`** defines a type.

The type is

`errno_t`

which is type **`int`**.

2.4 Common Definitions <stddef.h>

The header **`<stddef.h>`** defines a type.

The type is

`rsize_t`

which is the type **`size_t`**.[1]

2.5 Integer Types <stdint.H>

The header **`<stdint.h>`** defines a macro.

The macro is

`RSIZE_MAX`

which expands to a value[2] of type **`size_t`**. Functions that have parameters of type **`rsize_t`** consider it a runtime-constraint violation if the values of those parameters are greater than **`RSIZE_MAX`**.

Recommended Practice

Extremely large object sizes are frequently a sign that an object's size was calculated incorrectly. For example, negative numbers appear as very large positive numbers when converted to an unsigned type such as **`size_t`**. Also, some implementations do not support objects as large as the maximum value that can be represented by type **`size_t`**.

For those reasons, it is sometimes beneficial to restrict the range of object sizes to detect programming errors. For implementations targeting machines with large address spaces, it is recommended that **`RSIZE_MAX`** be defined as the smaller of the size of the largest object supported or **`(SIZE_MAX >> 1)`**, even if this limit is smaller than the size of some legitimate, but very large, objects. Implementations targeting machines with small address spaces

[1] See the description of the **`RSIZE_MAX`** macro in **`<stdint.h>`**.

[2] The macro **`RSIZE_MAX`** need not expand to a constant expression.

may wish to define **RSIZE_MAX** as **SIZE_MAX**, which means that no object size is considered a runtime-constraint violation.

2.6 Managed String Type <string.m.h>

The header **<string_m.h>** defines an abstract data type:

```
typedef void *string_m;
```

The structure referenced by this type is private and implementation defined. All managed strings of this type have a maximum string length that is determined when the string is created. For functions that have parameters of type **string_m**, it is a runtime-constraint violation if the maximum length of a managed string is exceeded.

Managed strings may also have a defined set of valid characters that can be used in the string. For functions that have parameters of type **string_m**, it is a runtime-constraint violation if a managed string contains invalid characters. For functions that have parameters of type **string_m** it is a runtime-constraint if the request requires allocating more memory than available.[3]

Managed strings support both null and empty strings. An empty string is a string that has zero characters. A null string is an uninitialized string, or a string that has been explicitly set to null.

For computing the length of a string to determine if the maximum length is exceeded, the length of a null-terminated byte string is the number of bytes, and the length of a wide character string is the number of characters. Thus, promoting a multi-byte null-terminated byte string may change its length.

2.7 General Utilities <stdlib.h>

The header **<stdlib.h>** defines six types.

The types are

errno_t

which is type **int**; and

rsize_t

[3] The library depends on **malloc()**and **realloc()**returning a null pointer to signify insufficient memory. On some systems, particularly systems using optimistic memory allocation schemes, **malloc()**may return a non-null pointer even when there is insufficient memory. On systems where there is no such mechanism to detect out-of-memory conditions, the library will not be able to properly validate this condition.

which is the type **`size_t`**; and

`constraint_handler_t`

which has the following definition

```
typedef void (*constraint_handler_t)(
    const char * restrict msg,
    void * restrict ptr,
    errno_t error);
```

and

```
malloc_handler_t
```

which has the following definition

```
typedef void * (*malloc_handler_t)(
    size_t size);
```

and

```
realloc_handler_t
```

which has the following definition

```
typedef void * (*realloc_handler_t)(
    void * ptr, size_t size);
```

and

```
free_handler_t
```

which has the following definition

```
typedef void (*free_handler_t)(void *ptr);
```

3 Library Functions

3.1 Utility Functions

3.1.1 The isnull_m Function

Synopsis

```
#include <string_m.h>
errno_t isnull_m(const string_m s, _Bool *nullstr);
```

Runtime-Constraints

`s` shall reference a valid managed string. `nullstr` shall not be a null pointer.

Description

The `isnull_m` function tests whether the managed string `s` is null and delivers this result in the parameter referenced by `nullstr`, given the managed string `s`.

Returns

The `isnull_m` function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.1.2 The isempty_m Function

Synopsis

```
#include <string_m.h>
errno_t isempty_m(const string_m s,
                  _Bool *emptystr);
```

Runtime-Constraints

`s` shall reference a valid managed string. `emptystr` shall not be a null pointer.

Description

The `isempty_m` function tests whether the managed string `s` is empty and delivers this result in the parameter referenced by `emptystr`, given the managed string `s`.

Returns

The **isempty_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.1.3 Creating a string_m

3.1.3.1 The strcreate_m Function

Synopsis

```
     #include <string_m.h>
     errno_t strcreate_m(string_m *s, const char *cstr,
                  const rsize_t maxlen, const char *charset);
```

Runtime-Constraints

s shall not be a null pointer. **charset** shall not be an empty string (denoted by **""**).

Description

The **strcreate_m** function creates a managed string, referenced by **s**, given a conventional string **cstr** (which may be null or empty). **maxlen** specifies the maximum length of the string in characters. If **maxlen** is zero, the system-defined maximum length is used. **charset** restricts the set of allowable characters to be those in the null-terminated byte string **cstr** (which may be empty). If **charset** is a null pointer, no restricted character set is defined. If specified, duplicate characters in a **charset** are ignored. Characters in the **charset** may be provided in any order. The **\0** character cannot be specified as part of **charset**.

Returns

The **strcreate_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.1.3.2 The wstrcreate_m Function

Synopsis

```
     #include <string_m.h>
     errno_t wstrcreate_m(string_m *s,
                   const wchar_t *wcstr,
                   const rsize_t maxlen,
                   const wchar_t *charset);
```

Runtime-Constraints

s shall not be a null pointer. **charset** shall not be an empty string (denoted by **L""**).

Description

The **wstrcreate_m** function creates a managed string, referenced by **s**, given a wide character string **wcstr** (which may be null or empty). **maxlen** specifies the maximum length of the string in characters. If **maxlen** is 0, the system-defined maximum length is used. **charset** restricts the set of allowable characters to be those in the wide character string **wcstr** (which may be empty). If **charset** is a null pointer, no restricted character set is defined. Characters in the **charset** may be provided in any order. The **\0** character cannot be specified as part of **charset**.

Returns

The **wstrcreate_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.1.4 The isntbs_m Function

Synopsis

```
#include <string_m.h>
errno_t isntbs_m(const string_m s,
                 _Bool *ntbstr);
```

Runtime-Constraints

s shall reference a valid managed string. **ntbstr** shall not be a null pointer.

Description

The **isntbs_m** function tests whether the managed string **s** is a null-terminated byte string and delivers this result in the parameter referenced by **ntbstr**, given the managed string **s**.

Returns

The **isntbs_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.1.5 The iswide_m Function

Synopsis

```
#include <string_m.h>
errno_t iswide_m(const string_m s,
                 _Bool *widestr);
```

Runtime-Constraints

s shall reference a valid managed string. **widestr** shall not be a null pointer.

Description

The **iswide_m** function tests whether the managed string **s** is a wide character string and delivers this result in the parameter referenced by **widestr**, given the managed string **s**.

Returns

The **iswide_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.1.6 The strdelete_m Function

Synopsis

```
     #include <string_m.h>
     errno_t strdelete_m(string_m *s);
```

Runtime-Constraints

s shall not be a null pointer. ***s** shall reference a valid managed string.

Description

The **strdelete_m** function deletes the managed string referenced by ***s** (which may be null or empty). **s** is set to a null pointer.

Returns

The **strdelete_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.1.7 The strlen_m Function

Synopsis

```
     #include <string_m.h>
     errno_t strlen_m(const string_m s, rsize_t *size);
```

Runtime-Constraints

s shall reference a valid managed string. **size** shall not be a null pointer.

Description

The **strlen_m** function computes the length of the managed string **s** and stores the result into the variable referenced by **size**. If the managed string is either null or empty, the length is computed as zero. For a null-terminated byte string, the length is the number of bytes. For a wide character string, the length is the number of characters.

Returns

The **strlen_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.1.8 Extracting a conventional string

3.1.8.1 The cgetstr_m Function

Synopsis

```
#include <string_m.h>
errno_t cgetstr_m(const string_m s, char **string);
```

Runtime-Constraints

s shall reference a valid managed string. **string** shall not be a null pointer. ***string** must be a null pointer.

Description

The **cgetstr_m** function allocates storage for, and returns a pointer to, a null-terminated byte string represented by the managed string **s** and referenced by **string**. The caller is responsible for freeing ***string** when the null-terminated byte string is no longer required.

Example

```
if (retValue = cgetstr_m(str1, &cstr)) {
  fprintf(stderr, "error %d from cgetstr_m.\n",
                                            retValue);
} else {
  printf("(%s)\n", cstr);
  free(cstr); // free duplicate string
}
```

Returns

The **cgetstr_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned. If there is a runtime-constraint violation, ***string** is set to a null pointer.

3.1.8.2 The wgetstr_m Function

Synopsis

```
#include <string_m.h>
errno_t wgetstr_m(const string_m s, wchar_t **wcstr);
```

Runtime-Constraints

s shall reference a valid managed string. **wcstr** shall not be a null pointer. ***wcstr** must be a null pointer.

Description

The **wgetstr_m** function delivers a wide character string into the variable referenced by **wcstr**, given the managed string **s**. The caller is responsible for freeing ***wcstr** when the wide character string is no longer required.

Returns

The **wgetstr_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned. If there is a runtime-constraint violation, ***wcstr** is set to a null pointer.

3.1.9 The strdup_m Function

Synopsis

```
#include <string_m.h>
errno_t strdup_m(string_m *s1, const string_m s2);
```

Runtime-Constraints

s1 shall not be a null pointer. **s2** shall reference a valid managed string.

Description

The **strdup_m** function creates a duplicate of the managed string **s2** and stores it in **s1**. The duplicate shall have the same set of valid characters and maximum length.

Returns

The **strdup_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.2 Copying Functions

3.2.1 Unbounded string copy

3.2.1.1 The strcpy_m Function

Synopsis

```
#include <string_m.h>
errno_t strcpy_m(string_m s1, const string_m s2);
```

Runtime-Constraints

s1 and **s2** shall reference valid managed strings.

Description

The **strcpy_m** function copies the managed string **s2** into the managed string **s1**. Note that the set of valid characters and maximum length are not copied, as these are attributes of **s1**.[4]

Returns

The **strcpy_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.2.1.2 The cstrcpy_m Function

Synopsis

```
     #include <string_m.h>
     errno_t cstrcpy_m(string_m s1, const char *cstr);
```

Runtime-Constraints

s1 shall reference a valid managed string.

Description

The **cstrcpy_m** function copies the string **cstr** into the managed string **s1**.

Returns

The **cstrcpy_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.2.1.3 The wstrcpy_m Function

Synopsis

```
     #include <string_m.h>
     errno_t wstrcpy_m(string_m s1,
                const wchar_t *wcstr);
```

Runtime-Constraints

s1 shall reference a valid managed string.

Description

The **wstrcpy_m** function copies the string **wcstr** into the managed string **s1**.

[4] If **s2** contains characters that are not in the set of valid characters or exceeds the maximum length defined for **s1**, a runtime constraint violation occurs as described in Section 2.6.

Returns

The **wstrcpy_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.2.2 The strncpy_m Function

Synopsis

```
     #include <string_m.h>
     errno_t strncpy_m (string_m s1,
                 const string_m s2,
                 rsize_t n);
```

Runtime-Constraints

s1 and **s2** shall reference valid managed strings.

Description

The **strncpy_m** function copies not more than **n** characters from the managed string **s2** to the managed string **s1**. If **s2** does not contain **n** characters, the entire string is copied. If **s2** contains more than **n** characters, **s1** is set to the string containing the first **n** characters.

Returns

The **strncpy_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.3 Concatenation Functions

3.3.1 Unbounded concatenation

3.3.1.1 The strcat_m Function

Synopsis

```
     #include <string_m.h>
     errno_t strcat_m(string_m s1, const string_m s2);
```

Runtime-Constraints

s1 and **s2** shall reference valid managed strings.

Description

The **strcat_m** function concatenates the managed string **s2** onto the end of the managed string **s1**.

Returns

The **strcat_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.3.1.2 The cstrcat_m Function

Synopsis

```
#include <string_m.h>
errno_t cstrcat_m(string_m s, const char *cstr);
```

Runtime-Constraints

s shall reference a valid managed string.

Description

The **cstrcat_m** function concatenates the null-terminated byte string **cstr** onto the end of the managed string **s**. If **cstr** is a null pointer, this function returns without modifying **s**.

Returns

The **cstrcat_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.3.1.3 The wstrcat_m Function

Synopsis

```
#include <string_m.h>
errno_t wstrcat_m(string_m s,
          const wchar_t *wcstr);
```

Runtime-Constraints

s shall reference a valid managed string. **wcstr** shall not be a null pointer.

Description

The **wstrcat_m** function concatenates the wide character string **wcstr** onto the end of the managed string **s**. If **wcstr** is a null pointer, this function returns without modifying **s**.

Returns

The **wstrcat_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.3.2 Bounded concatenation

3.3.2.1 The strncat_m Function

Synopsis

```
     #include <string_m.h>
     errno_t strncat_m (string_m s1,
                 const string_m s2,
                 rsize_t n);
```

Runtime-Constraints

s1 and **s2** shall reference valid managed strings.

Description

The **strncat_m** function appends not more than **n** characters from the managed string **s2** to the end of the managed string **s1**. If s2 is a null pointer, **strncat_m** returns without modifying **s1**.

Returns

The **strncat_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.3.2.2 The cstrncat_m Function

Synopsis

```
     #include <string_m.h>
     errno_t cstrncat_m (string_m s,
                 const char *cstr,
                 rsize_t n);
```

Runtime-Constraints

s shall reference a valid managed string.

Description

The **cstrncat_m** function appends not more than **n** bytes from the null-terminated byte string **cstr** to the end of the managed string **s**. If **cstr** is null, **cstrncat_m** returns without modifying **s**. The **cstrncat_m** function guarantees that the resulting string **s** is properly terminated.

Returns

The **cstrncat_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.3.2.3 The wstrncat_m Function

Synopsis

```
     #include <string_m.h>
     errno_t wstrncat_m (string_m s,
                 const wchar_t *wcstr,
                 rsize_t n);
```

Runtime-Constraints

s shall reference a valid managed string.

Description

The **wstrncat_m** function appends not more than **n** characters from the wide character string **wcstr** to the end of the managed string **s**. If **wcstr** is a null pointer, the **wstrncat_m** function returns without modifying **s**. The **wstrncat_m** function guarantees that the resulting string **s** is properly terminated.

Returns

The **wstrncat_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.4 Comparison Functions

The sign of a non-zero value delivered by the comparison functions **strcmp_m**, and **strncmp_m** is determined by the sign of the difference between the values of the first pair of characters (both interpreted as **unsigned char** but promoted to **int**) that differ in the objects being compared.

For the purpose of comparison, a null string is less than any other string including an empty string. Null strings are equal and empty strings are equal.

The set of valid characters defined for each string is not a factor in the evaluation although it is held as an invariant that each managed string contains only characters identified as valid for that string.

3.4.1 Unbounded comparison

3.4.1.1 The strcmp_m Function

Synopsis

```
     #include <string_m.h>
     errno_t strcmp_m (const string_m s1,
                 const string_m s2,
                 int *cmp);
```

Runtime-Constraints

s1 and **s2** shall reference valid managed strings. **cmp** shall not be a null pointer.

Description

The **strcmp_m** function compares the managed string **s1** to the managed string **s2** and sets **cmp** to an integer value greater than, equal to, or less than zero, accordingly as **s1** is greater than, equal to, or less than **s2.**

Returns

The **strcmp_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.4.1.2 The cstrcmp_m Function

Synopsis

```
     #include <string_m.h>
     errno_t cstrcmp_m (const string_m s1,
                 const char *cstr,
                 int *cmp);
```

Runtime-Constraints

s1 shall reference a valid managed string. **cmp** shall not be a null pointer.

Description

The **cstrcmp_m** function compares the managed string **s1** to the null-terminated byte string **cstr** and sets **cmp** to an integer value greater than, equal to, or less than zero, accordingly as **s1** is greater than, equal to, or less than **cstr.**

Returns

The **cstrcmp_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.4.1.3 The wstrcmp_m Function

Synopsis

```
     #include <string_m.h>
     errno_t wstrcmp_m (const string_m s1,
                 const wchar_t *wstr,
                 int *cmp);
```

Runtime-Constraints

s1 shall reference a valid managed string. **cmp** shall not be a null pointer.

Description

The **wstrcmp_m** function compares the managed string **s1** to the wide character string **wstr** and sets **cmp** to an integer value greater than, equal to, or less than zero, accordingly as **s1** is greater than, equal to, or less than **wstr.**

Returns

The **wstrcmp_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.4.2 The strcoll_m Function

Synopsis

```
#include <string_m.h>
errno_t strcoll_m (const string_m s1,
             const string_m s2,
             int *cmp);
```

Runtime-Constraints

s1 and **s2** shall reference valid managed strings. **cmp** shall not be a null pointer.

Description

The **strcoll_m** function compares the managed string **s1** to the managed string **s2**, both interpreted as appropriate to the **LC_COLLATE** category of the current locale. The **strcoll_m** function then sets **cmp** to an integer value greater than, equal to, or less than zero, accordingly as **s1** is greater than, equal to, or less than **s2** when both are interpreted as appropriate to the current locale.

Returns

The **strcoll_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.4.3 Bounded string comparison

3.4.3.1 The strncmp_m Function

Synopsis

```
#include <string_m.h>
errno_t strncmp_m (const string_m s1,
             const string_m s2,rsize_t n,
             int *cmp);
```

Runtime-Constraints

s1 and **s2** shall reference valid managed strings. **cmp** shall not be a null pointer.

Description

The **strncmp_m** function compares not more than **n** characters from the managed string **s1** to the managed string **s2** and sets **cmp** to an integer value greater than, equal to, or less than zero, accordingly as **s1** is greater than, equal to, or less than **s2**. If **n** is equal to 0, the **strncmp_m** function sets **cmp** to the integer value zero, regardless of the contents of the string.

Returns

The **strncmp_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.4.3.2 The cstrncmp_m Function

Synopsis

```
     #include <string_m.h>
     errno_t cstrncmp_m (const string_m s1,
                 const char *cstr, rsize_t n,
                 int *cmp);
```

Runtime-Constraints

s1 shall reference a valid managed string. **cmp** shall not be a null pointer.

Description

The **cstrncmp_m** function compares not more than **n** bytes (bytes that follow a null character are not compared) from the managed string **s1** to the null-terminated byte string **cstr** and sets **cmp** to an integer value greater than, equal to, or less than zero, accordingly as **s1** is greater than, equal to, or less than **cstr.** If **n** is equal to 0, the **cstrncmp_m** function sets **cmp** to the integer value zero, regardless of the contents of the string.

Returns

The **cstrncmp_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.4.3.3 The wstrncmp_m Function

Synopsis

```
     #include <string_m.h>
     errno_t wstrncmp_m (const string_m s1,
                 const wchar_t *wstr, rsize_t n,
                 int *cmp);
```

Runtime-Constraints

s1 shall reference a valid managed string. **cmp** shall not be a null pointer.

Description

The **`wstrncmp_m`** function compares not more than **`n`** characters (characters that follow a null character are not compared) from managed string **`s1`** to the wide character string **`wstr`** and sets **`cmp`** to an integer value greater than, equal to, or less than zero, accordingly as **`s1`** is greater than, equal to, or less than **`wstr.`** If **`n`** is equal to zero, the **`wstrncmp_m`** function sets **`cmp`** to the integer value zero, regardless of the contents of the string.

Returns

The **`wstrncmp_m`** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.5 Search Functions

3.5.1 The strtok_m Function

Synopsis

```
#include <string_m.h>
errno_t strtok_m(string_m token, string_m str,
            const string_m delim, string_m ptr);
```

Runtime-Constraints

`token`, **`str`**, **`delim`**, and **`ptr`** shall reference valid managed strings.

Description

The **`strtok_m`** function scans the managed string **`str`**. The substring of **`str`** up to but not including the first occurrence of any of the characters contained in the managed string **`delim`** is returned as the managed string **`token`**. The remainder of the managed string **`str`** (after but not including the first character found from **`delim`**) is returned as the managed string **`ptr`**. If **`str`** does not contain any characters in **`delim`** (or if **`delim`** is either empty or null), **`token`** shall be set to **`str,`** and **`ptr`** will be set to the null string.

Returns

The **`strtok_m`** function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

3.5.2 The cstrchr_m Function

Synopsis

```
#include <string_m.h>
errno_t cstrchr_m(string_m out, const string_m str,
            char c);
```

Runtime-Constraints

out and **str** shall reference valid managed strings.

Description

The **cstrchr_m** function scans the managed string **str** for the first occurrence of **c**. **out** is set to the string containing and following the first occurrence of **c**. If **str** does not contain **c**, **out** is set to the null string.

Returns

The **cstrchr_m** function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

3.5.3 The wstrchr_m Function

Synopsis

```
#include <string_m.h>
errno_t wstrchr_m(string_m out, const string_m str,
            wchar_t wc);
```

Runtime-Constraints

out and **str** shall reference valid managed strings.

Description

The **wstrchr_m** function scans the managed string **str** for the first occurrence of **wc**. **out** is set to the string containing and following the first occurrence of **wc**. If **str** does not contain **wc**, **out** is set to the null string.

Returns

The **wstrchr_m** function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

3.5.4 The strspn_m Function

Synopsis

```
#include <string_m.h>
errno_t strspn_m(string_m str, string_m accept,
            rsize_t *len);
```

Runtime-Constraints

str and **accept** shall reference a valid managed string. **len** shall not be a null pointer.

Description

The **`strspn_m`** function computes the length of the maximum initial segment of the managed string **`str`** which consists entirely of characters from the managed string **`accept`**. The **`strspn_m`** function sets **`*len`** to this length. If the managed string **`str`** is null or empty, **`*len`** is set to zero.

Returns

The **`strspn_m`** function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

3.5.5 The strcspn_m Function

Synopsis

```
#include <string_m.h>
errno_t strcspn_m(string_m str, string_m reject,
            rsize_t *len);
```

Runtime-Constraints

`str` and **`accept`** shall reference valid managed strings. **`len`** shall not be a null pointer.

Description

The **`strcspn_m`** function computes the length of the maximum initial segment of the managed string **`str`** , which consists entirely of characters *not* from the managed string **`reject`**. The **`strcspn_m`** function sets **`*len`** to this length. If the managed string **`str`** is null or empty **`*len`** is set to zero. If the managed string **`reject`** is null or empty, **`*len`** is set to the length of **`str`**.

Returns

The **`strcspn_m`** function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

3.6 Configuration Functions

3.6.1 The setcharset_m Function

Synopsis

```
#include <string_m.h>
errno_t setcharset_m(string_m s,
           const string_m charset);
```

Runtime-Constraints

`s` shall reference a valid managed string.

Description

The **`setcharset_m`** function sets the subset of allowable characters to be those in the managed string **`charset`** (which may be null or empty). If **`charset`** is a null pointer or the managed string represented by **`charset`** is null, a restricted subset of valid characters is not enforced. If the managed string **`charset`** is empty, then only empty or null strings can be created.

Returns

The **`setcharset_m`** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.6.2 The setmaxlen_m Function

Synopsis

```
     #include <string_m.h>
     errno_t setmaxlen_m(string_m s, rsize_t maxlen);
```

Runtime-Constraints

`s` shall reference a valid managed string.

Description

The **`setmaxlen_m`** function sets the maximum length of the managed string **`s`**. If **`maxlen`** is 0, the system-defined maximum length is used.

Returns

The **`setmaxlen_m`** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.6.3 The setallocators_m Function

Synopsis

```
     #include <string_m.h>
     errno_t  setallocators_m _m(malloc_handler_t mh,
                 realloc_handler_t rh, free_handler_t fh);
```

Runtime-Constraints

`mh`, **`rh`**, and **`fh`** shall not be a null pointer and shall point to valid functions.

Description

The **setallocators_m** function sets the memory allocation functions used by the managed string library. If not explicitly set, **mh** defaults to **malloc()**, **rh** defaults to **realloc()**, and **fh** defaults to **free()**.

Returns

The **setallocators_m** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

3.7 printf-derived Functions

These functions are the managed string equivalents to the **printf-**derived functions in C.

The '**%s**' specification refers to a managed string, rather than a null-terminated byte string or wide character string. The format specification **'%ls'** indicates that the managed string should be output as a wide character string. The format specification **'%hs'** indicates that the managed string should be output as a null-terminated byte string. All **printf**-derived functions will output a null-terminated byte string if (1) any specified output stream is byte oriented and (2) the format string and all argument strings are null-terminated byte strings; otherwise the output will be a wide-character string.

Applying a byte output function to a wide-oriented stream or a wide character output function to a byte-oriented stream will result in a runtime-constraint error.

The **'%n'** specification is not recognized.

3.7.1 The sprintf_m Function

Synopsis

```
     #include <string_m.h>
     errno_t sprintf_m(string_m buf, const string_m fmt, int
                  *count, ...);
```

Runtime-Constraints

buf and **fmt** shall reference valid managed strings. The managed string **fmt** shall be a valid format compatible with the arguments after **fmt**.

Description

The **sprintf_m** function formats its parameters after the third parameter into a string according to the format contained in the managed string **fmt** and stores the result in the managed string **buf**.

If not a null pointer, **`*count`** is set to the number of characters written in **`buf`**, not including the terminating null character.

Returns

The **`sprintf_m`** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.7.2 The vsprintf_m Function

Synopsis

```
     #include <string_m.h>
     errno_t vsprintf_m(string_m buf,
                 const string_m fmt,
                 int *count,
                 va_list args);
```

Runtime-Constraints

`buf` and **`fmt`** shall reference a valid managed string. **`fmt`** shall not be a null pointer. The managed string **`fmt`** shall be a valid format compatible with the arguments **`args`**.

Description

The **`vsprintf_m`** function formats its parameters **`args`** into a string according to the format contained in the managed string **`fmt`** and stores the result in the managed string **`buf`**.

If not a null pointer, **`*count`** is set to the number of characters written in **`buf`**, not including the terminating null character.

Returns

The **`vsprintf_m`** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.7.3 The snprintf_m Function

Synopsis

```
     #include <string_m.h>
     errno_t snprintf_m(string_m buf, int max,
                const string_m fmt, int *count, ...);
```

Runtime-Constraints

`buf` and **`fmt`** shall reference a valid managed string. **`fmt`** shall not be a null pointer. The managed string **`fmt`** shall be a valid format compatible with the arguments after **`fmt`**.

Description

The **snprintf_m** function formats its parameters after the fourth parameter into a string according to the format contained in the managed string **fmt** and stores the result in the managed string **buf**. If the resulting string contains more than **max** characters, it is truncated.

If not a null pointer, ***count** is set to the number of characters that would have been written had **max** been sufficiently large, not counting the terminating null character. Thus, the output will be completely written if and only if the returned value is nonnegative and less than **max**.

Returns

The **snprintf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.7.4 The vsnprintf_m Function

Synopsis

```
#include <string_m.h>
errno_t vsnprintf_m(string_m buf, int max,
              const string_m fmt, int *count,
              va_list args);
```

Runtime-Constraints

Buf and **fmt** shall reference a valid managed string. **fmt** shall not be a null pointer. The managed string **fmt** shall be a valid format compatible with the arguments **args**.

Description

The **vsprintf_m** function formats its parameters **args** into a string according to the format contained in the managed string **fmt** and stores the result in the managed string **buf**. If the resulting string contains more than **max** characters, it is truncated.

If not a null pointer, ***count** is set to the number of characters that would have been written had **max** been sufficiently large, not counting the terminating null character. Thus, the output will be completely written if and only if the returned value is nonnegative and less than **max**.

Returns

The **vsprintf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.7.5 The printf_m Function

Synopsis

```
#include <string_m.h>
errno_t printf_m(const string_m fmt, int *count, ...);
```

Runtime-Constraints

fmt shall reference a valid managed string. **fmt** shall not be a null pointer. The managed string **fmt** shall be a valid format compatible with the arguments after **fmt**.

Description

The **printf_m** function formats its parameters after the second parameter into a string according to the format contained in the managed string **fmt** and outputs the result to standard output.

If not a null pointer, ***count** is set to the number of characters transmitted.

Returns

The **printf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.7.6 The vprintf_m Function

Synopsis

```
#include <string_m.h>
errno_t vprintf_m(const string_m fmt, int *count,
            va_list args);
```

Runtime-Constraints

fmt shall reference a valid managed string. **fmt** shall not be a null pointer. The managed string **fmt** shall be a valid format compatible with the arguments **args**.

Description

The **vprintf_m** function formats its parameters **args** into a string according to the format contained in the managed string **fmt** and outputs the result to standard output.

If not a null pointer, ***count** is set to the number of characters transmitted.

Returns

The **vprintf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.7.7 The fprintf_m Function

Synopsis

```
#include <string_m.h>
errno_t fprintf_m(FILE *file, const string_m fmt, int
            *count, ...);
```

Runtime-Constraints

fmt shall reference a valid managed string. **fmt** shall not be a null pointer. The managed string **fmt** shall be a valid format compatible with the arguments after **fmt**. **file** shall not be a null pointer.

If not a null pointer, ***count** is set to the number of characters transmitted.

Description

The **fprintf_m** function formats its parameters after the third parameter into a string according to the format contained in the managed string **fmt** and outputs the result to **file**.

Returns

The **fprintf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.7.8 The vfprintf_m Function

Synopsis

```
     #include <string_m.h>
     errno_t vfprintf_m(FILE *file, const string_m fmt,
                  int *count, va_list args);
```

Runtime-Constraints

fmt shall reference a valid managed string. **fmt** shall not be a null pointer. The managed string **fmt** shall be a valid format compatible with the arguments **args**. **file** shall not be a null pointer.

Description

The **vfprintf_m** function formats its parameters **args** into a string according to the format contained in the managed string **fmt** and outputs the result to **file**.

If not a null pointer, ***count** is set to the number of characters transmitted.

Returns

The **vfprintf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.8 scanf-derived Functions

These functions are the managed string equivalents to the **scanf**-derived functions in C. Managed string format strings differ from standard C format strings primarily in that they are represented as managed strings. The **'%s'** specification refers to a managed string, rather than a null-terminated byte string or wide character string. The use of **`char*`** or **`wchar_t*`** pointers in the **`varargs`** list will result in a runtime-constraint if detected. The managed string read by **'`%s`'** is created as a null-terminated byte string if the input string is a null-terminated byte string or the input stream has byte orientation; otherwise a wide character string is created. The format specification **'`%ls`'** indicates that the managed string should be created as a wide character string. The format specification **'`%hs`'** indicates that the managed string should be created as a null-terminated byte string.

Applying a byte input function to a wide-oriented stream or a wide character input function to a byte-oriented stream will result in a runtime-constraint error.

3.8.1 The sscanf_m Function

Synopsis

```
     #include <string_m.h>
     errno_t sscanf_m(string_m buf, const string_m fmt,
                 int *count, ...);
```

Runtime-Constraints

`buf` and **`fmt`** shall reference a valid managed string. **`fmt`** shall not be a null pointer. The managed string **`fmt`** shall be a valid format compatible with the arguments after **`fmt`**.

Description

The **sscanf_m** function processes the managed string **buf** according to the format contained in the managed string **fmt** and stores the results using the arguments after **count**.

If not a null pointer, ***count** is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Returns

The **sscanf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.8.2 The vsscanf_m Function

Synopsis

```
#include <string_m.h>
errno_t vsscanf_m(string_m buf,
            const string_m fmt,
            int *count,
            va_list args);
```

Runtime-Constraints

buf and **fmt** shall reference a valid managed string. **fmt** shall not be a null pointer. The managed string **fmt** shall be a valid format compatible with the arguments **args**.

Description

The **vsscanf_m** function processes the managed string **buf** according to the format contained in the managed string **fmt** and stores the results using the arguments in **args**.

If not a null pointer, ***count** is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Returns

The **vsscanf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.8.3 The scanf_m Function

Synopsis

```
#include <string_m.h>
errno_t scanf_m(const string_m fmt, int *count, ...);
```

Runtime-Constraints

fmt shall reference a valid managed string. **fmt** shall not be a null pointer. The managed string **fmt** shall be a valid format compatible with the arguments after **count**.

Description

The **scanf_m** function processes input from standard input according to the format contained in the managed string **fmt** and stores the results using the arguments after **count**.

If not null, ***count** is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Returns

The **scanf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.8.4 The vscanf_m Function

Synopsis

```
#include <string_m.h>
errno_t vscanf_m(const string_m fmt, int *count,
                 va_list args);
```

Runtime-Constraints

fmt shall reference a valid managed string. **fmt** shall not be a null pointer. The managed string **fmt** shall be a valid format compatible with the arguments **args**.

Description

The **vscanf_m** function processes input from standard input according to the format contained in the managed string **fmt** and stores the results using the arguments in **args**.

If not null, ***count** is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Returns

The **vscanf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.8.5 The fscanf_m Function

Synopsis

```
     #include <string_m.h>
     errno_t fscanf_m(FILE *file, const string_m fmt,
                int *count, ...);
```

Runtime-Constraints

fmt shall reference a valid managed string. **fmt** shall not be a null pointer. The managed string **fmt** shall be a valid format compatible with the arguments after **count**.
file shall not be a null pointer.

Description

The **fscanf_m** function processes input from **file** according to the format contained in the managed string **fmt** and stores the results using the arguments after **count**.

If not a null pointer, ***count** is set to the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

Returns

The **fscanf_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.9 String Slices

3.9.1 The strslice_m Function

Synopsis

```
     #include <string_m.h>
     errno_t strslice_m(string_m s1,
                const string_m s2,
                rsize_t offset, rsize_t len);
```

Runtime-Constraints

s1 and **s2** shall reference valid managed strings. There shall be sufficient memory to store the result.

Description

The **strslice_m** function takes up to **len** characters from **s2**, starting at the **offset** character in the string and stores the result in **s1**. If there are insufficient characters to copy **len** characters, all available characters are copied. If **offset** is greater than the number of characters in **s2**, **s1** is set to the null string. If **offset** is equal to the number of characters in **s2** or **len** is 0, **s1** is set to the empty string.

Returns

The **strslice_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.9.2 The strleft_m Function

Synopsis

```
     #include <string_m.h>
     errno_t strleft_m(string_m s1,
                 const string_m s2,
                 rsize_t len);
```

Runtime-Constraints

s1 and **s2** shall reference valid managed strings. There shall be sufficient memory to store the result.

Description

The **strleft_m** function copies up to **len** characters from the start of the managed string **s2** to the managed string **s1**. If **s2** does not have **len** characters, the entire string is copied. If **s2** is a null string, **s1** is set to the null string.

Returns

The **strleft_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.9.3 The strright_m Function

Synopsis

```
     #include <string_m.h>
     errno_t strleft_m(string_m s1,
                 const string_m s2,
                 rsize_t len);
```

Runtime-Constraints

s1 and **s2** shall reference valid managed strings. There shall be sufficient memory to store the result.

Description

The **strright_m** function copies up to the last **len** characters from the managed string **s2** to the managed string **s1**. If **s2** does not have **len** characters, the entire string is copied. If s2 is a null string, **s1** is set to the null string.

Returns

The **strright_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.9.4 The cchar_m Function

Synopsis

```
#include <string_m.h>
errno_t cchar_m(const string_m s,
               rsize_t offset,
               char *c);
```

Runtime-Constraints

s shall reference a valid managed string. **c** shall not be a null pointer. **offset** shall be less than the length of the managed string **s**. The character to be returned in **c** shall be representable as a **char**.

Description

The **cchar_m** function sets **c** to the **offset** character (the first character having an **offset** of 0) in the managed string **s**.

Returns

The **cchar_m** function returns zero if no runtime-constraints were violated.
Otherwise, a non-zero value is returned.

3.9.5 The wchar_m Function

Synopsis

```
#include <string_m.h>
errno_t wchar_m(const string_m s,
               rsize_t offset,
               wchar_t *wc);
```

Runtime-Constraints

s1 shall reference a valid managed string. **wc** shall not be a null pointer. **offset** shall be less than the length of the managed string **s1**.

Description

The **wchar_m** function sets **wc** to the **offset** character (the first character having an **offset** of 0) in the managed string **s**.

Returns

The **`wchar_m`** function returns zero if no runtime-constraints were violated. Otherwise, a non-zero value is returned.

4 Reference

URL is valid as of the publication date of this document.

[ISO/IEC 99] International Organization for Standardization, International Electrotechnical Commission. ISO/IEC 9899:1999, *Programming Languages—C.* http://www.open-std.org/JTC1/SC22/WG14 /www/docs/n1124.pdf (May 6, 2005).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE May 2006	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE

Specifications for Managed Strings

5. FUNDING NUMBERS

FA8721-05-C-0003

6. AUTHOR(S)

Hal Burch, Fred Long, Robert Seacord

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER

CMU/SEI-2006-TR-06

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ESC-TR-2006-006

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified/Unlimited, DTIC, NTIS

12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This report describes a managed string library for the C programming language. Many software vulnerabilities in C programs result from the misuse of standard C string manipulation functions. Programming errors common to string manipulation logic include buffer overflow, truncation errors, string termination errors, and improper data sanitation. The managed string library provides mechanisms to eliminate or mitigate these problems and improve system security. A proof-of-concept implementation of the managed string library is available from the Secure Coding area of the CERT Web site.

14. SUBJECT TERMS

managed strings, C program language, specification, string library, ISO/IEC 9899:1999

15. NUMBER OF PAGES

4949

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL